

**《数字逻辑电路》实验报告**[文档副标题]



**综合实验：MIPS计算机系统**

**姓名：林伟&陈佳伟**

**学号：161220078 161220009**

**16级计算机科学与技术系2班**

**邮箱：**[2386556428@qq.com](mailto:2386556428@qq.com)

**时间：2017/12/30**

1. **实验目的**

本实验的目标是在Nexys4开发板上实现一个简单的计算机系统，能够运行简单的指令，并处理一定量的输入输出。在所有功能开发完毕后，希望能够完成基本的terminal功能，即键盘输入命令，并在显示器上输出结果。

1. **实验原理（背景知识）**

**PartI**

**MIPS指令集：**该指令集是经典的RISC指令集，实现起来较为简单。MIPS 32的所有指令长度都是32bit，分为三种基本类型：

* R-Type：含3个寄存器操作数
* I-Type：2个寄存器操作数，及一个16bit立即数操作数
* J-Type：跳转，26bit立即数操作数

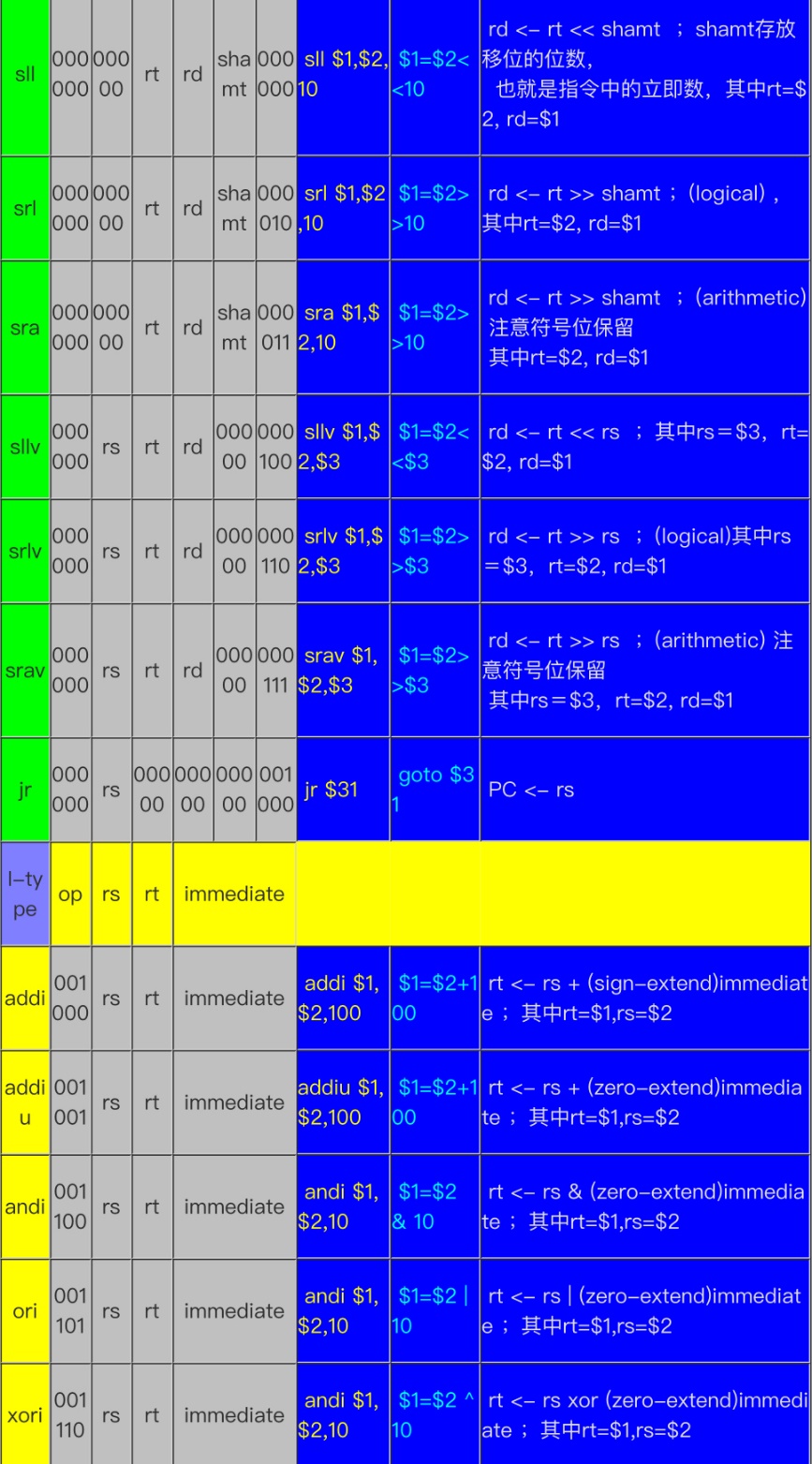
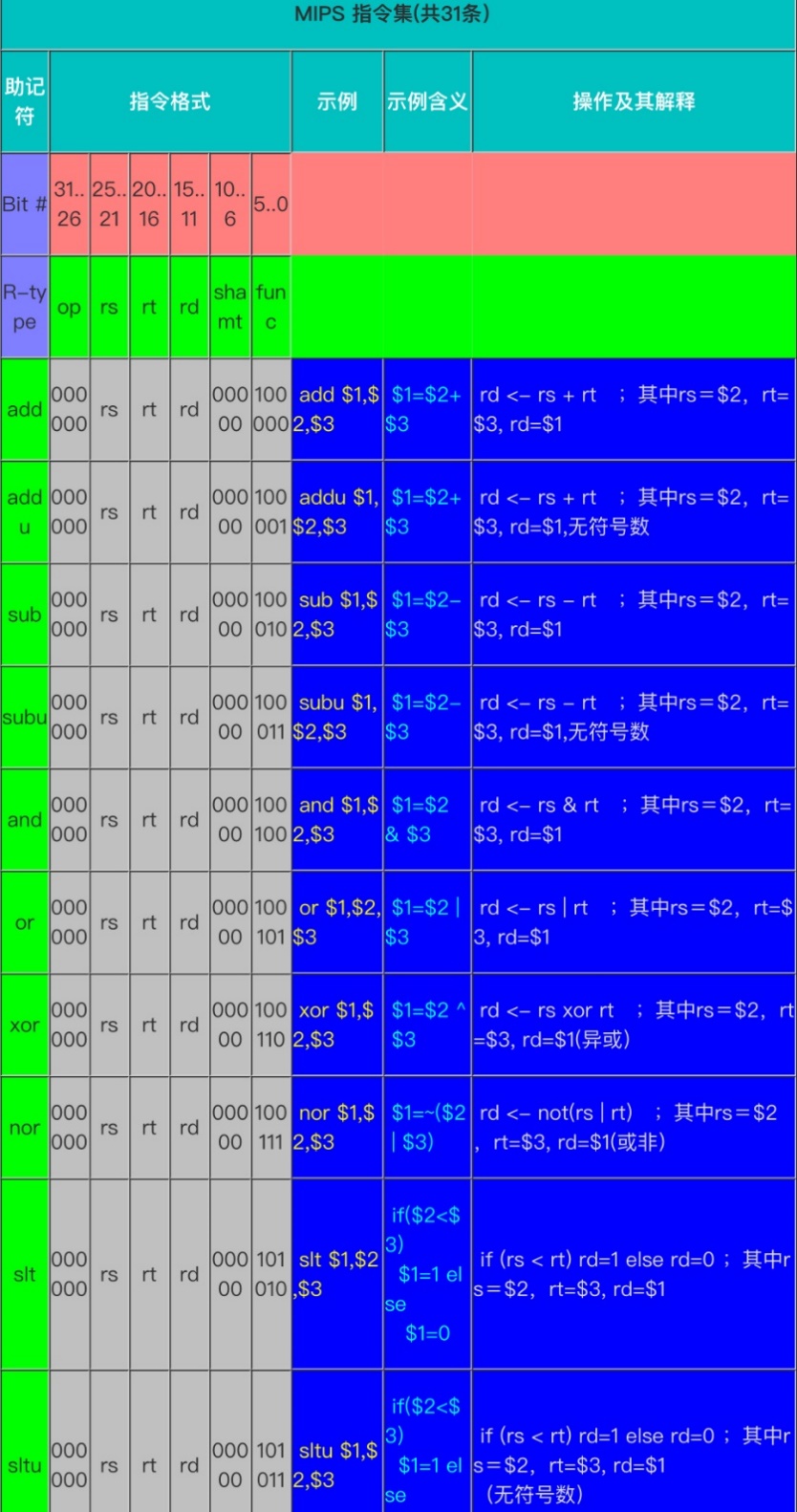
格式如下：

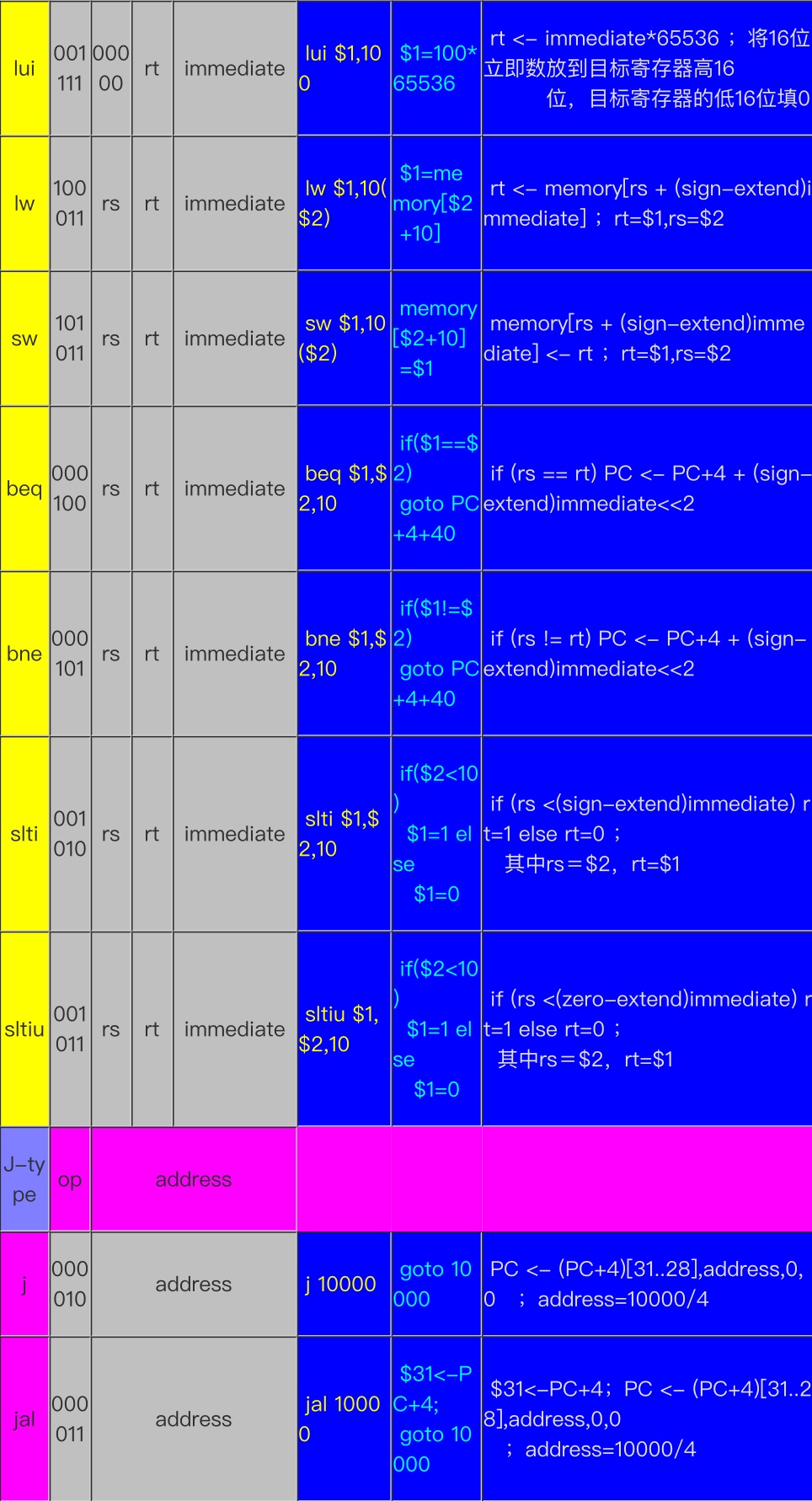
../../../Computer%20Architecture/beamer%20slides/02_ISA_Principles/mipsisaa.pdf

其中opcode必定为指令前6bit，源、目的寄存器也都在特定位置出现，所以指令解码非常方便。其中包含以下几类指令：

* ALU指令：可以是2寄存器操作数，结果送入目的寄存器，或一个寄存器一个立即数，结果送入目的寄存器。包括有无符号的加减法，移位，及逻辑操作等等。
* Load Store指令：一般是寄存器+立即数偏移量，读取/写入内存。**注意：**MIPS中所有数据都需要先load进入寄存器才能进行操作，不能像x86一样直接对内存数据进行算术处理。
* 分支与跳转指令：条件分支包括BEQ，BNE等等，根据寄存器内容选择是否跳转（没有flag寄存器）。无条件跳转是26位立即数。JAL用于函数调用，自动将返回地址放入r31寄存器。**注意：**MIPS中有跳转延迟槽的概念，跳转指令下一条指令不管是否跳转都会执行。所以，建议在所有跳转指令后加上一条NOP(全0，add r0,r0,r0)。

MIPS 32共32个32bit的寄存器（5 bit寄存器地址），其中寄存器r0中的内容总是0，寄存器r31中存储函数调用的返回地址（在JAL指令中实现）。在本实现中一共实现了33条指令：





以及bgtz指令和自己设计的屏幕输出指令。

（2）**VGA的工作原理**

图像的显示是以像素（点）为单位，显示器的分辨率是指屏幕每行有多少个像素及每帧有多少行，标准的VGA分辨率是640×480，也有更高的分辨率，如1024×768、1280×1024、1920×1200等。从人眼的视觉效果考虑，屏幕刷新的频率（每秒钟显示的帧数）应该大于24，这样屏幕看起来才不会闪烁，VGA显示器一般的刷新频率是60HZ。

每一帧图像的显示都是从屏幕的左上角开始一行一行进行的，行同步信号是一个负脉冲，行同步信号有效后，由RGB端送出当前行显示的各像素点的RGB电压值，当一帧显示结束后，由帧同步信号送出一个负脉冲，重新开始从屏幕的左上端开始显示下一帧图像，如图2所示。



图 2 显示器扫描示意图

RGB端并不是所有时间都在传送像素信息，由于CRT的电子束从上一行的行尾到下一行的行头需要时间，从屏幕的右下角回到左上角开始下一帧也需要时间，这时RGB送的电压值为0（黑色），这些时间称为电子束的行消隐时间和场消隐时间，行消隐时间以像素为单位，帧消隐时间以行为单位。VGA行扫描、场扫描时序示意图如图3所示：

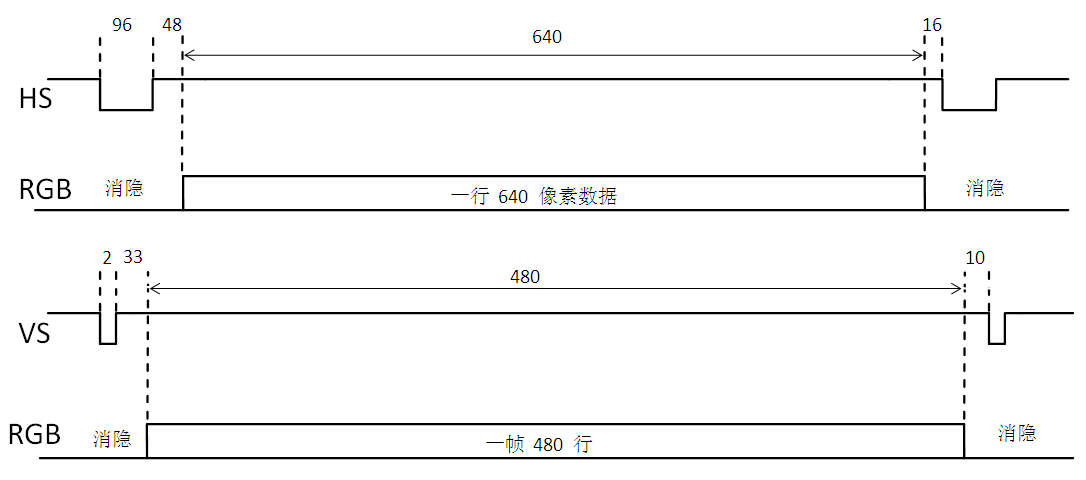


图3 VGA行扫描、场扫描时序示意图

如上图所示，有效地显示一行信号需要96+48+640+16=800个像素点的时间，其中行同步负脉冲宽度为96个像素点时间，行消隐后沿需要48个像素点时间，每行显示640个像素点，行消隐前沿需要16个像素点的时间，一行显示时间为640个像素点时间，一行消隐时间为160个像素点时间。

有效显示一帧图像需要2+33+480+10=525行时间，其中场同步负脉冲宽度为2个行显示时间，场消隐后沿需要33个行显示时间，每场显示480行，场消隐前沿需要10个行显示时间，一帧显示时间为525行显示时间，一帧消隐时间为45行显示时间。

1. **实验器材/环境**

**vivado2016.4+NEXYS4**

1. **实验设计思路（验收实验）**

实验大致分为两个部分，第一个是内核软件部分，即cpu操作，数据处理都在这个部分完成，第二个部分是显示部分，显示器部分需要显示的是当前执行的汇编代码，以及最后的结果和运行过程中的寄存器值。其中第一部分由林伟（161220078）完成，第二部分由陈佳伟（161220009）完成。首先是第一部分设计的展示：

Part I

【1】关于指令集：使用的是MIPS 32位指令集

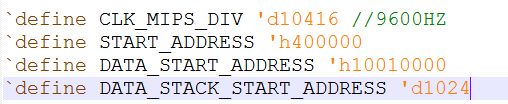
总体设计思路：

首先需要设计32个32位的寄存器以及一定大小的栈空间。寄存器直接用32个寄存器组即可，栈空间使用同步的储存器。然后设计出ALU运算模块。最后因为不支持变量移位，所以我们还要添加一个移位运算模块。最后根据各条指令设计出完整的CPU。

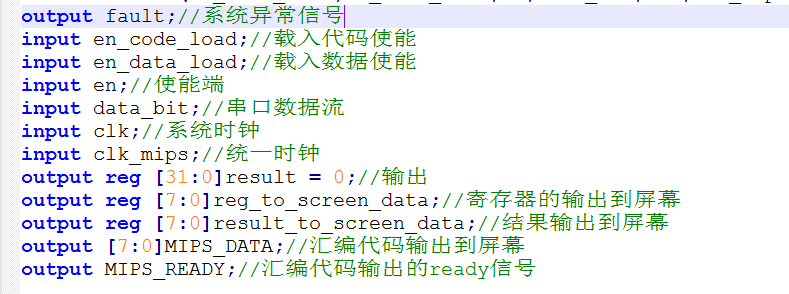


各个模块设计思路：

首先为了方便，我先定义了几个宏：

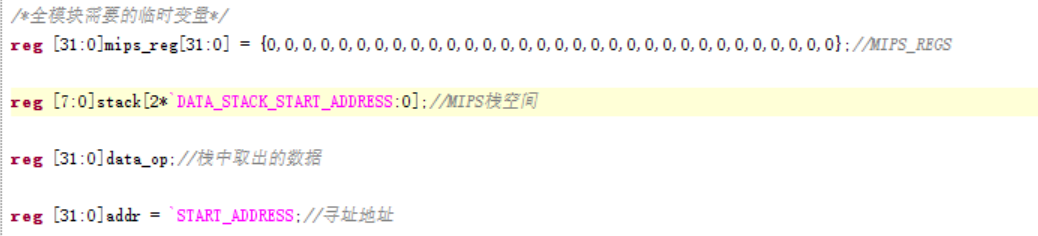


第一个是整个系统的运转频率，方便进行调整，START\_ADDRESS是代码段的逻辑地址，即汇编模拟器直接给出的地址。DATA\_START\_ADDRESS是数据的逻辑开始地址，也是由汇编模拟器直接给出的。第三个是DATA\_STACK\_START\_ADDRESS是数据的物理地址起始处。



这是解码执行模块的接口：第一个是系统异常信号，出现了未知的指令码等。

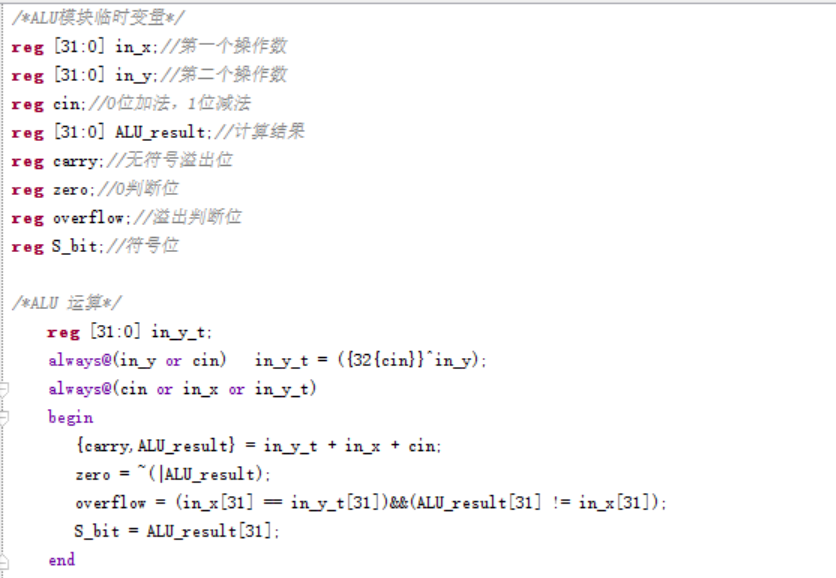
首先是整体的模块需要用到的寄存器和栈空间：



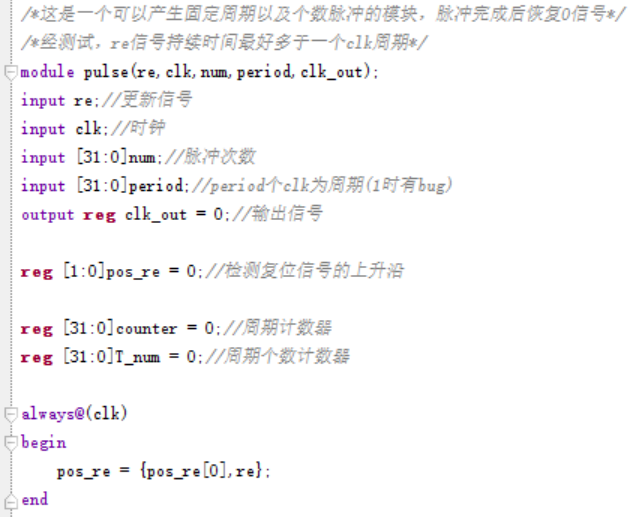
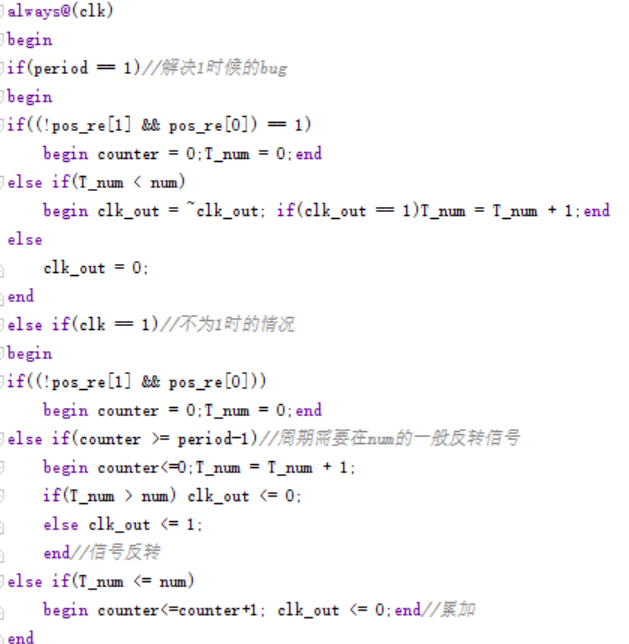
[1]这是同步栈空间的读写模块，在模块中实现代码段以及数据段的载入以及数据的读写功能。（采用同步的意义在于减少LUT寄存器的使用，在开发板中LUT资源是比较紧张的，如果采用异步的话，能够支持的空间就会很小了）



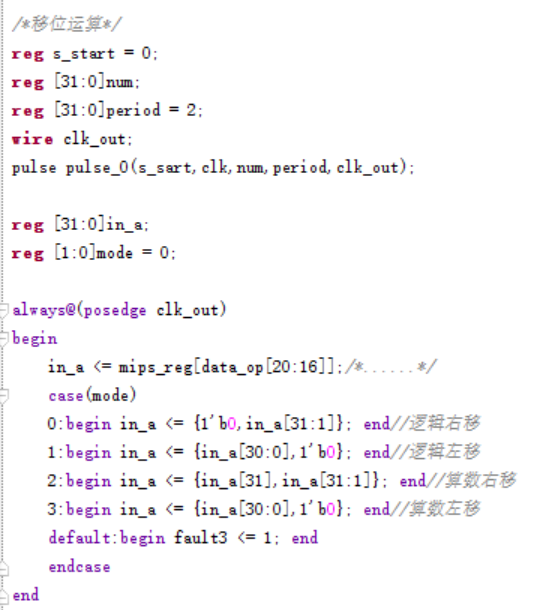
[2]接下来是ALU运算模块ALU运算模块的数据输入在时钟的上升沿，计算结果的获取在时钟的下降沿，以确保数据不会发生冲突：



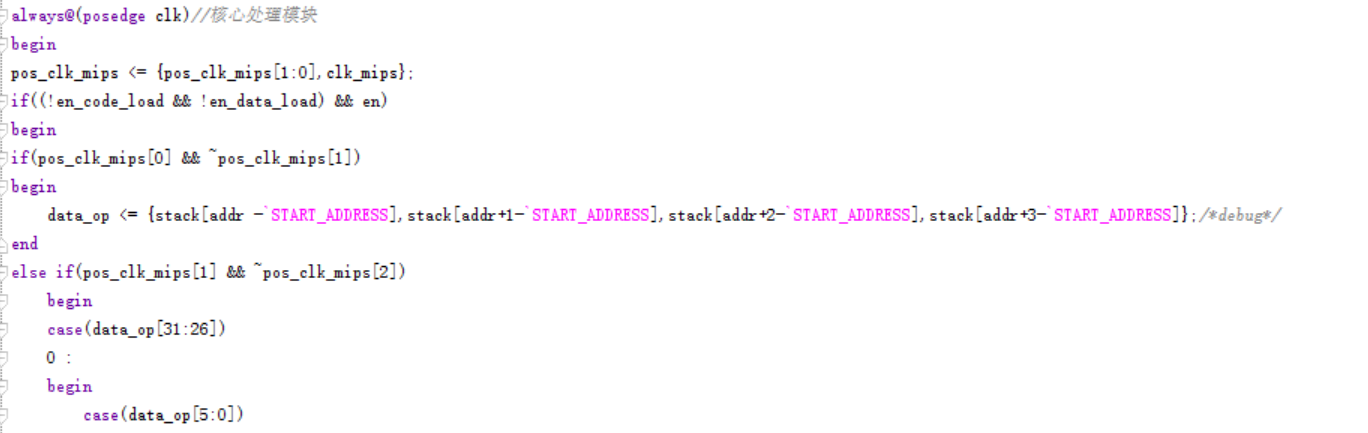
[3]为了实现移位运算模块，再次之前实现一个脉冲的模块：这是一个可以产生固定周期以及个数脉冲的模块，脉冲完成后恢复0信号，有了此模块，我们就可以实现移位n位的操作—即在每次上升沿时向左或者向右位移一位。

[4]移位模块:

移位模块需要在在操作中给出需要给出需要位移的次数，以及位移的操作，接收到开始信号后，脉冲模块就会不断发出num个脉冲帮助完成位移操作。

[5]核心解码模块：

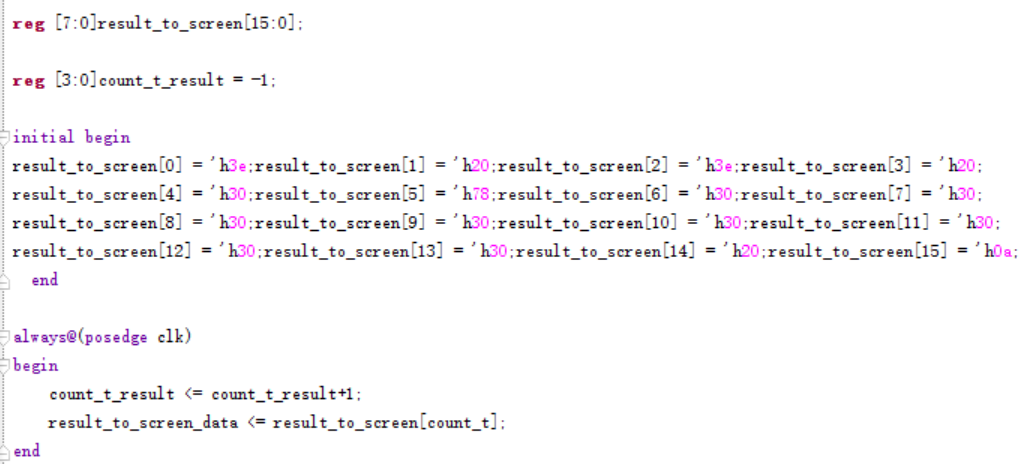


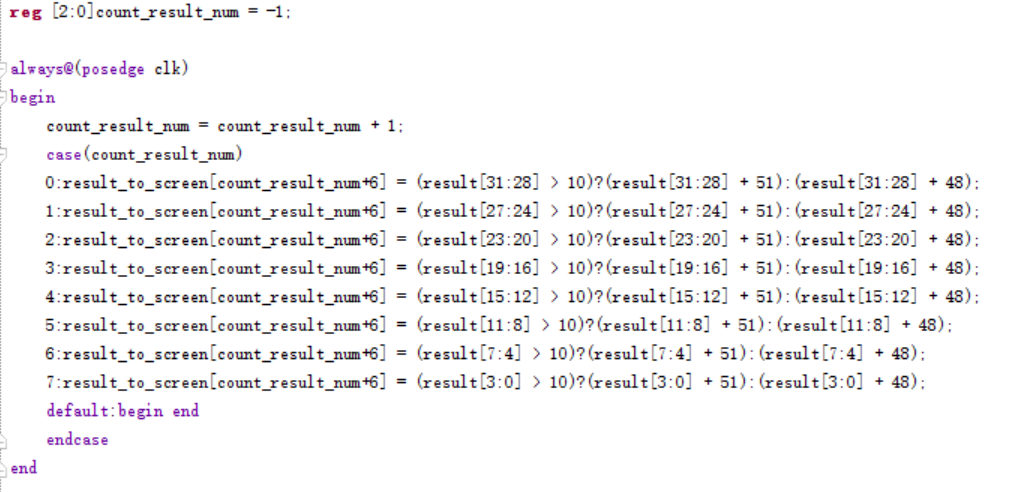
根据data\_op操作码，我们对操作码进行解码（具体解码代码为case语句，过长就不全贴在此处了），比如我们得到的指令码是3c011001，则通过解码我们获得其26-31位为：15，对应case为上升沿不需要数据对接：

下降沿：根据16-20位给出的目的寄存器完成立即数的赋值，即完成了装在寄存器的任务。

[6]外设对接：

以向屏幕输出运算结果为例：

我们设置了缓冲区，因为需要输出对应的ASCII码值，所以我们需要进行转换，转换规则为：如果这个4bite的数字是0-9的话加上48（48是数字0的ASCII值），如果是a-f的话加上51（61是字母a的ASCII值）即可。然后在每个时钟沿向显示器发送一个8bit的数据。显示器缓冲区接收数据。

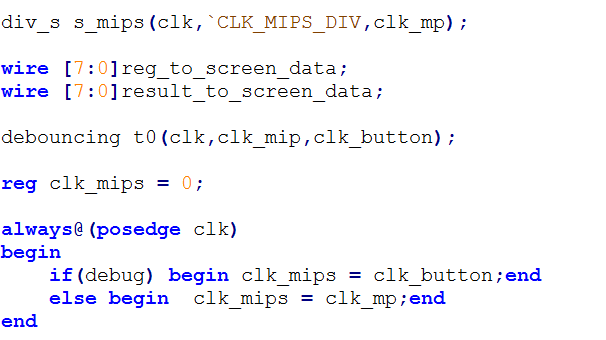


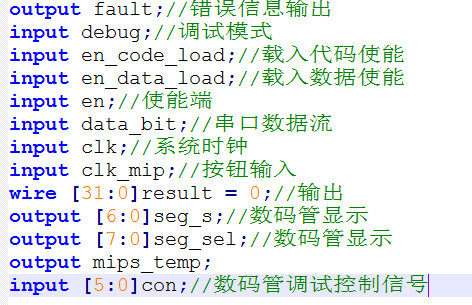
[7]数码管调试模块：

为了调试代码，我们设计了数码管的输出以方便调试，数码管可以通过开关拨动来显示当前地址、操作码、各寄存器、以及部分栈空间的值：



[8]顶层模块：

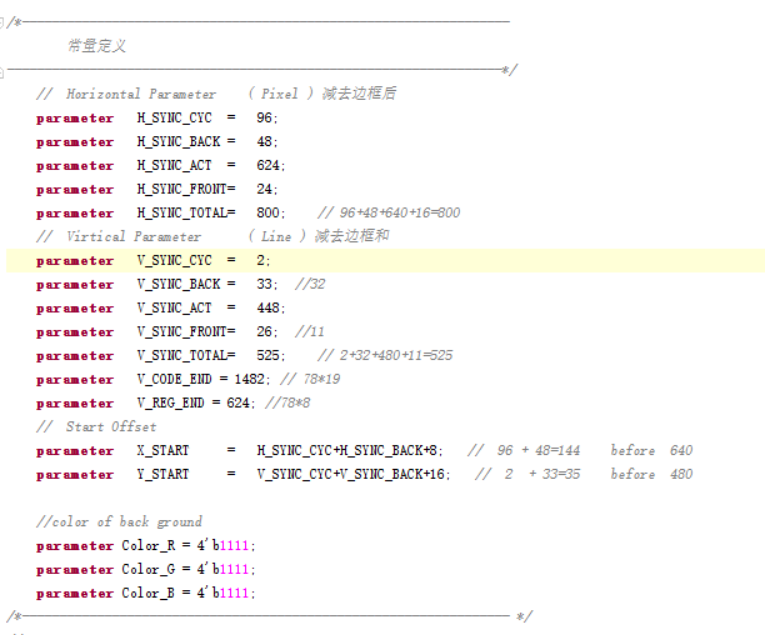
根据debug信号决定单步调试还是系统时钟自动执行，单步调试信号由用户按下按钮输入，系统时钟有分频器给出，最高支持100Mhz/64的频率，因为移位操作有可能需要移位32位，需要32个时钟周期，则限制了速度

顶层模块接口

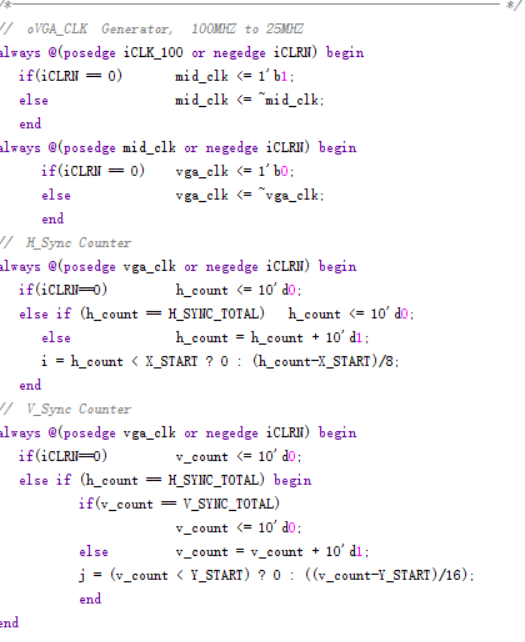
Part II

VGA显示部分

1. 代码框架

实验中VGA显示部分使用的整体框架是在VGA实验时使用的静态显示框架，其常量定义如下图。

VGA屏幕遍历的代码如下，通过行指针h\_count 和 帧指针v\_count来确定VGA显示屏上的位置。



1. 主要内容

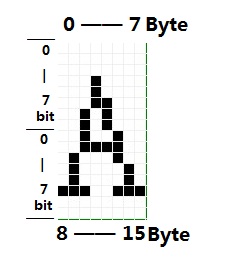
本次实验中，我们需要在显示屏上显示三块内容：汇编代码、输出数据、寄存器值；

整体上我们将显示屏分为两个大块：函数执行块（显示汇编代码以及函数执行的输出数据）和函数调试块（显示寄存器值）。

1. 字库寄存器

要实现在显示器上的字符（字母以及数字）显示，我们需要每个字符在屏幕上对应的字库数据，生成方式如下：

使用字模提取V2.1软件，设置字体为宋体、12，纵向取模，字节倒序（即高位在下）。这些设置可以根据实际情况设置。用C51格式生成字模，大小是8\*16，每个字符用16个字节表示。如字符A的显示如下：



取模数据为：

0x00,0x00,0xC0,0x38,0xE0,0x00,0x00,0x00,0x20,0x3C,0x23,0x02,0x02,0x27,0x38,0x20,

以此方法得到所以字符的字库数据以后，将其整理为一个可用于初始化字符字库存储器的初始化.txt文件即可，经初始化之后的ASCII字库存储器字符下标即为其ASCII码减去0x20（可显示的字符的第一个，空格的ASCII码）。

1. 字符在显示器上的显示思路——屏幕分块：

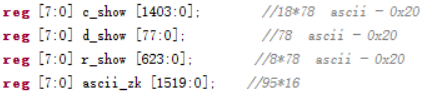
由字库数据决定的每个字符在显示器屏幕上占据的大小为八行十六列（8\*16），所以我们吧整个屏幕分为30行、80列，即每行可显示80个字符，每列可显示30个字符（后续增加边框后更改为每行可显示78个字符，一共有27行可用来显示字符）。这里用到了行和列的指针i，j来指示行列的位置坐标，行和列都是按顺序显示，所以位置坐标i，j也是直接由h\_count/v\_count直接转化而来。

屏幕分块之后，可直接将字符的字库数据填入屏幕对应块的像素点里（每个位对应一个像素点），能够很方便的实现字符的显示。

实验中将待显示的字符的ASCII码减去0x20的值存入显示存储器show中，这样通过一定的数学公式转换，即可实现通过数组show的值来寻找字库数据。

数组show分为三个部分（汇编代码段、函数结果数据段、寄存器数据段）如下，三个部分之和覆盖整个屏幕。

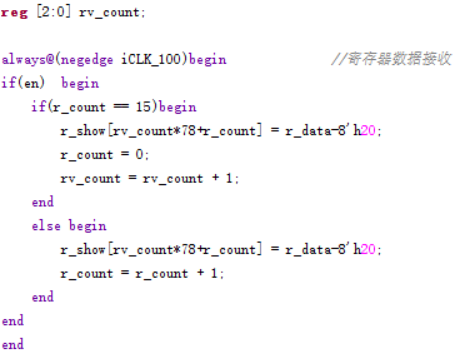
汇编代码段包括十八行，函数结果段只包含一行，而寄存器段包括八行。



1. 数据的接收

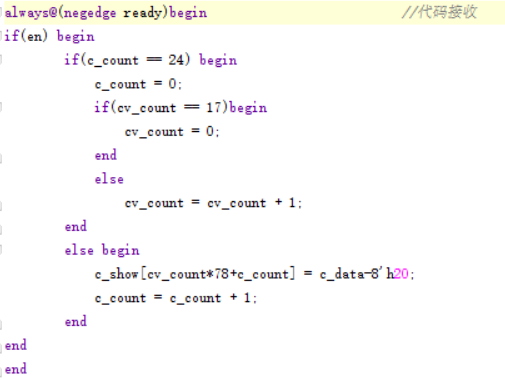
由于show的数据要持续不断的更新，所以设置使能端，数据接收使用最简单的相同时钟频率（100MHZ），只要使能端有效就一直接收数据。

函数调试模块主要显示八个寄存器的值，需要持续更新。

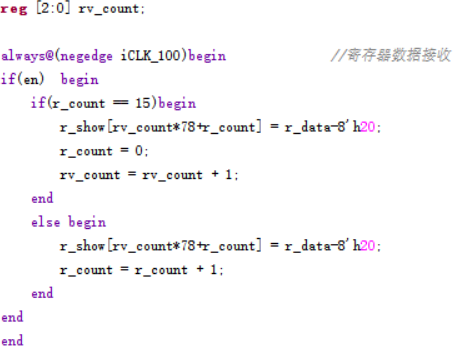


为避免数据传输的复杂性，实验中行对齐时减少了空格的计算过程，传输过来的数据被存放在指定的show的位置上（行前后均对齐），所以由rv\_count控制行，每当存到指定位置（如第16位）时就换行（以下的cv\_count、rv\_count与之功能相同）。

汇编代码段显示的数据由串口直接输送，所以以串口接收信号为时钟即可。



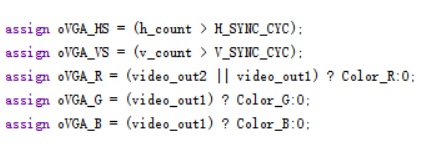
寄存器数据段直接接受内核传送的数据，使能端为真时开始持续接收。



1. 字符显示以及边框

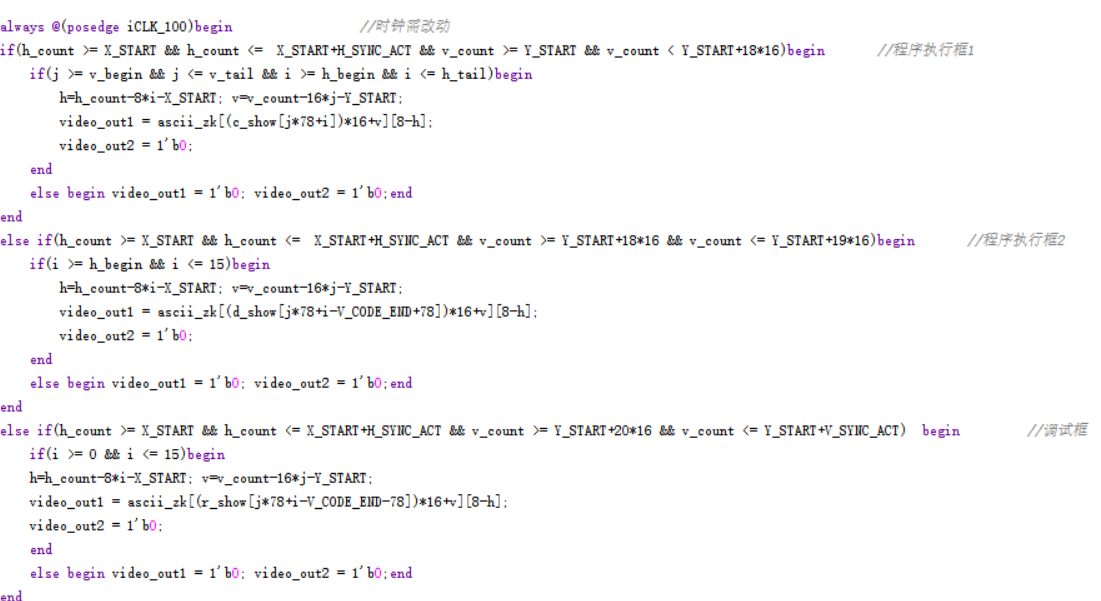
通过上面的分块思路、字符字库显示以及整体的显示数组show的方法，得到以下显示字符以及边框的代码：

video\_out1、video\_out2为颜色赋值信号。

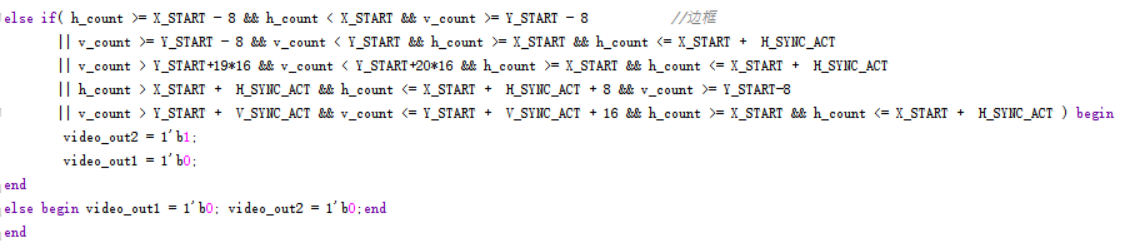


video\_out1给红、蓝、绿三种颜色等比例赋值，使得字符能够显示为白色。

字符的显示即为前面已经提到的，通过i，j确定坐标，取出对应坐标处的show的值，再到ascii\_zk数组中找到对应的字库数据，直接赋给video\_out1，即可在屏幕上得到对应字符的形状。



video\_out2在边框的位置只给红色赋值，使得边框显示为红色。

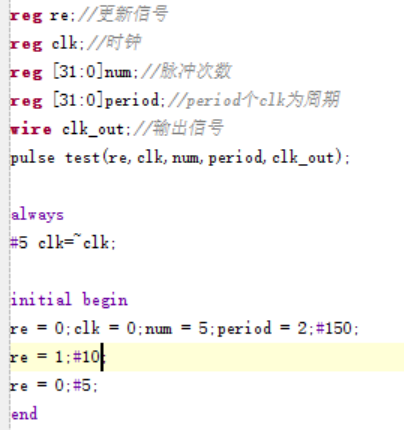


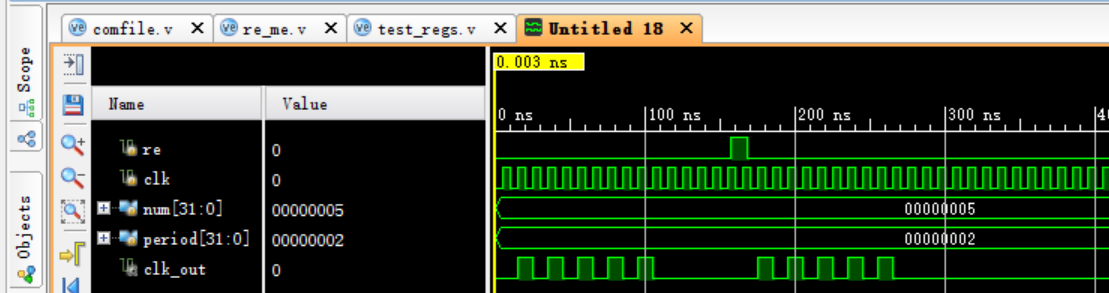
1. **实验的测试序列或验证方法**

**Part I**

主要对脉冲模块以及移位模块进行了仿真测试：

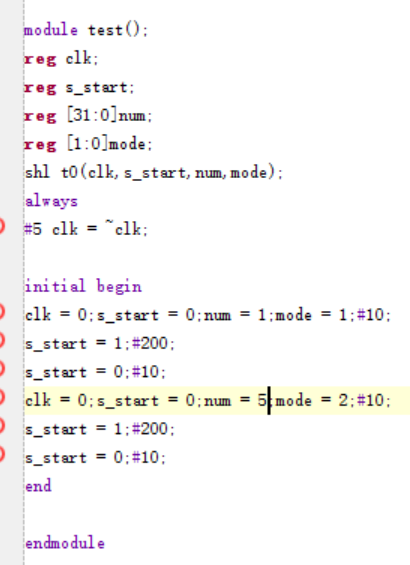
[1]脉冲模块：

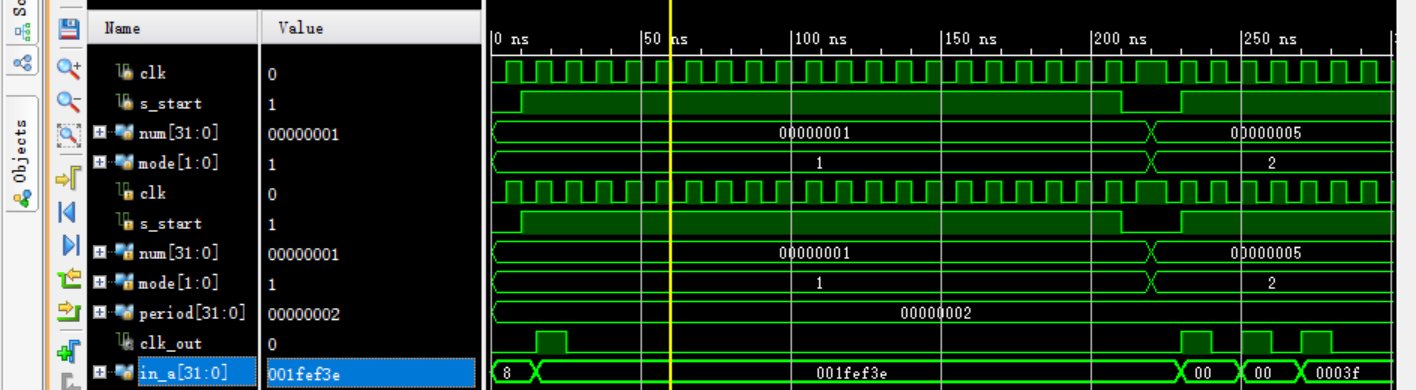
这是测试代码，测试能否发出固定个以及固定周期的脉冲信号，结果如下：



我们可以看到发出5个信号后就停止，知道下一个start信号来临

[2]移位运算：

测试移位结果是否正确：



可以看到移位运算的结果是正确的

#### PartII

由于VGA显示涉及的数据量过于巨大，所以在与内核对接以前，主要是通过将接收数据段接到开发板的开关上，手动输入数据来验证是否能正确的显示字符以及边框等的外观效果改善。

1. **实验过程（主要指验收实验）**
2. **实验结果、结论等**

实验结果是执行了计算斐波那契数列的代码，运算结果显示到屏幕上显示是正确的。

1. **实验中遇到的问题及解决方案**

**Part I**

在模块中，一开始使用的异步读写的栈空间，结果在没有语法错误的情况下不能生成.bit文件，原因是异步读写需要使用开发板的LUT资源，而这种资源是很有限的所以不能生成，改成同步的之后就成功生成了。

1. **实验的启示/意见和建议**